La programmation par contraintes a besoin d'analyse en moyenne

Charlotte Truchet

Equipe TASC
LS2N, UMR6004, Nantes
Université de Nantes

GdT Aléa 2018

Travaux en collaboration avec Xavier Lorca, Danièle Gardy, Giovanni Lo Bianco, Alejandro Arbelaez, Philippe Codognet.

Programmation par contraintes

Un *Constraint Satisfaction Problem* est donné par

- un ensemble de variables $V_1 \dots V_n$ (n fixé),
- un ensemble de domaines $D_1 \dots D_n$,
  où D_i représente les valeurs que la variable V_i peut prendre,

  NB : dans la suite, les domaines sont finis - ce n'est pas toujours le cas.
- un ensemble de contraintes $C_1 \dots C_p$, qui sont des relations logiques entre les variables.

Une solution du problème est une instanciation de valeurs des domaines aux variables, qui satisfait les contraintes.

Programmation par contraintes

Remarques :

- Une contrainte restreint les valeurs que les variables peuvent prendre. C'est dans le cadre classique la seule chose qui relie les variables.
- Par nature, un CSP définit un espace de possibilités $D_1 \times ... \times D_n$, de taille d^n (si $\forall i, |D_i| = d$).
- En général, on utilise la programmation par contraintes sur des problèmes NP-durs. Cela dit, ce n'est pas obligatoire.

Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
  $V_1 + 7 = V_3$,
  $V_1 * V_3 < 10$
  $\sum_i V_i < M$

Contraintes

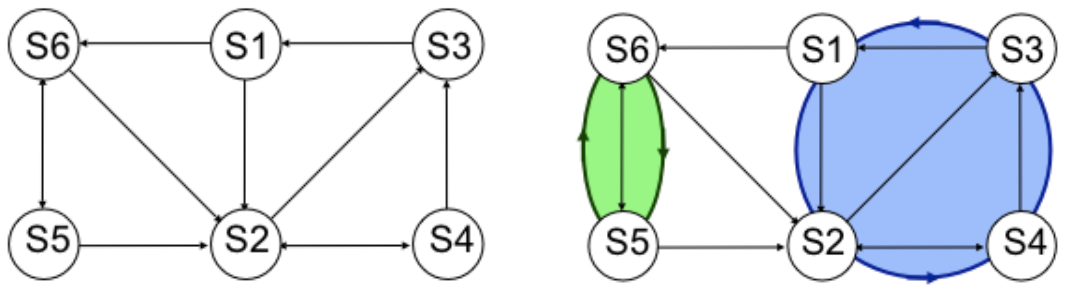
Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :

Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...



Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...

Contraintes

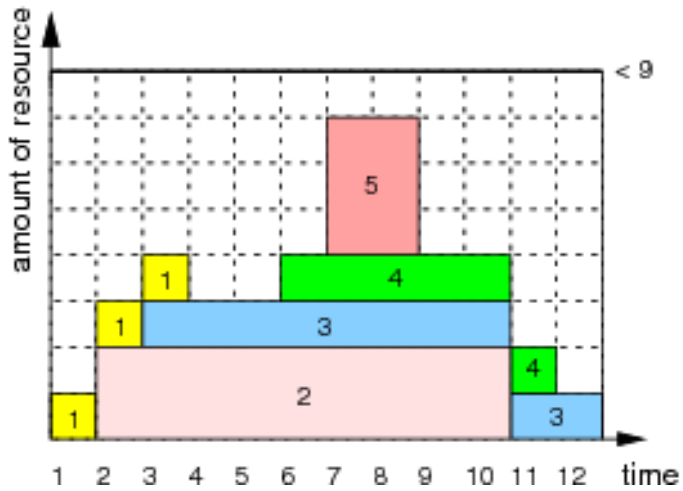
Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...

Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...
  - spécifiques : `cumulative`



Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...
  - spécifiques : `cumulative`
  - de cardinalité : `alldifferent`, `nvalue`, `atleast`, `gcc`...

Contraintes

Les langages de contraintes incluent en général

- expressions arithmétiques, fonctions "raisonnables",
- comparateurs usuels : $<, \leq, >, \geq, =, \neq$,
- contraintes globales :
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`...
  - spécifiques : `cumulative`
  - de cardinalité : `alldifferent`, `nvalue`, `atleast`, `gcc`...

Presque toutes les contraintes globales sont référencées dans le *Global Constraint Catalog*, avec un format commun, et les références bibliographiques.
`http://sofdem.github.io/gccat/`

Contraintes de cardinalité

`alldifferent`: les valeurs des variables sont toutes différentes.

utile pour : des salles (edt), certaines ressources (planification)...

`nvalue`,`atleast`, `atmost` : les variables prennent exactement (au plus, au moins) *N* valeurs différentes

utile pour : problèmes d'emploi du temps, frequency allocation...

NB : satisfiaisabilité et consistance sont NP-dures.

`gcc` : le nombre de chaque valeur pouvant être prise par les variables est dans un intervalle donné.

utile pour : problèmes d'emploi du temps, séries magiques...

Résolution

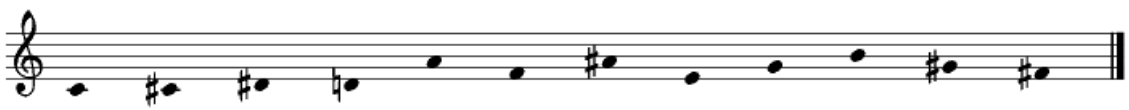
Deux familles de méthodes de résolution:

- méthodes complètes: exploration exhaustive de l'espace de recherche avec des méthodes pour éliminer rapidement les parties inutiles (consistance, propagation), dessin
- méthodes incomplètes: suite d'explorations locales de l'espace de recherche avec une boussole indiquant si on se rapproche d'une solution (fonctions de coût).

Applications

- musique, graphisme, code créatif,

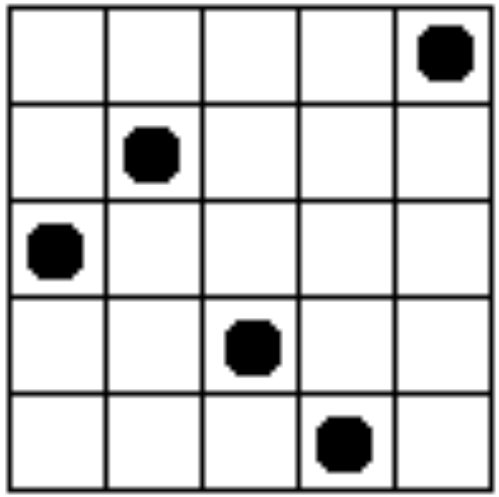
All-interval series



Applications

- musique, graphisme, code créatif,
- problèmes combinatoires "type IA",

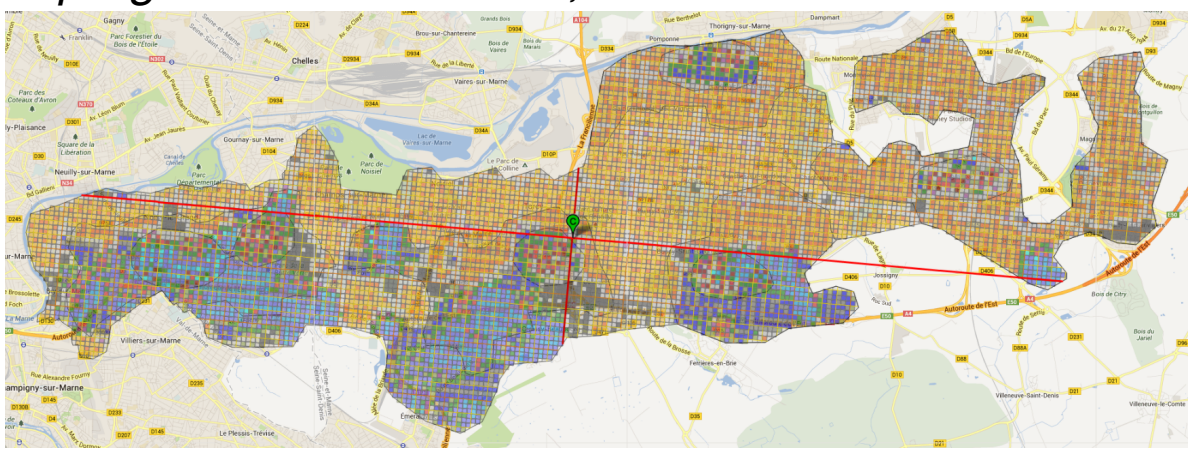
Costas Arrays



Applications

- musique, graphisme, code créatif,
- problèmes combinatoires "type IA",
- applications dédiées : cryptanalyse, médecine, football, urbanisme...

Pré-programmation urbaine, ici testée sur Marne la Vallée



Applications

- musique, graphisme, code créatif,
- problèmes combinatoires "type IA",
- applications dédiées : cryptanalyse, médecine, football, urbanisme...
- scheduling, scheduling avec une variante, scheduling avec une autre variante, etc.

Consistances

Une contrainte $C(X_1 ... X_n)$ est generalized arc-consistent (GAC) pour des domaines $D_1 ... D_n$ ssi pour toute variable X_i, pour toute valeur $v^i \in D_i$, il existe des valeurs $v^1 \in D_1, ..., v^{i-1} \in D_{i-1}, v^{i+1} \in D_{i+1}, ..., v^n \in D_n$ telles que $C(v^1, ... v^n)$.

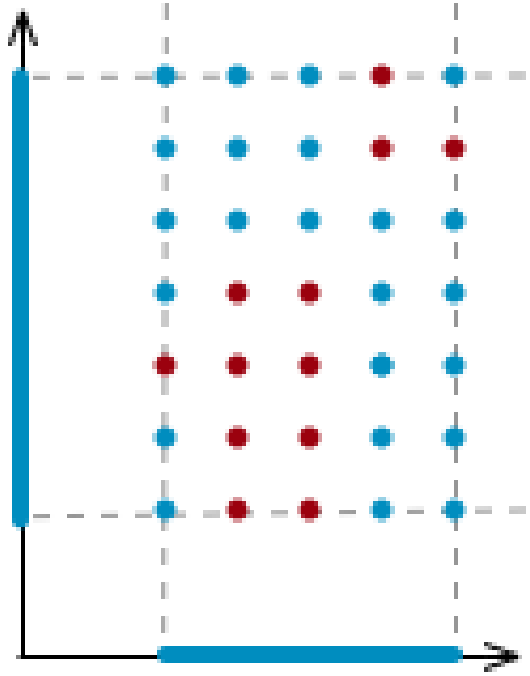
Consistances

Une contrainte $C(X_1 ... X_n)$ est generalized arc-consistent (GAC) pour des domaines $D_1 ... D_n$ ssi pour toute variable X_i, pour toute valeur $v^i \in D_i$, il existe des valeurs $v^1 \in D_1, ..., v^{i-1} \in D_{i-1}, v^{i+1} \in D_{i+1}, ..., v^n \in D_n$ telles que $C(v^1, ... v^n)$.

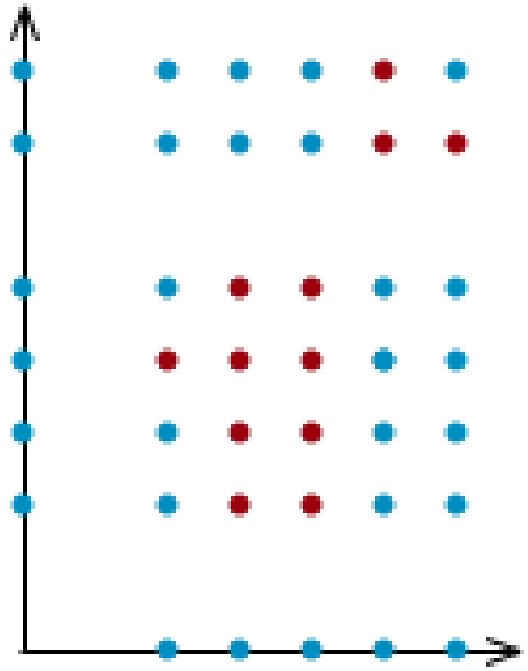
Une contrainte $C(X_1 ... X_n)$ est bound-consistent (BC) pour des domaines $D_1 ... D_n$ ssi les bornes de tous les domaines sont consistantes (au sens ci-dessus).

Consistances

En rouge les solutions, en bleu les domaines consistants.



Bound-consistency



Generalized arc-consistency

Consistances

Remarques :

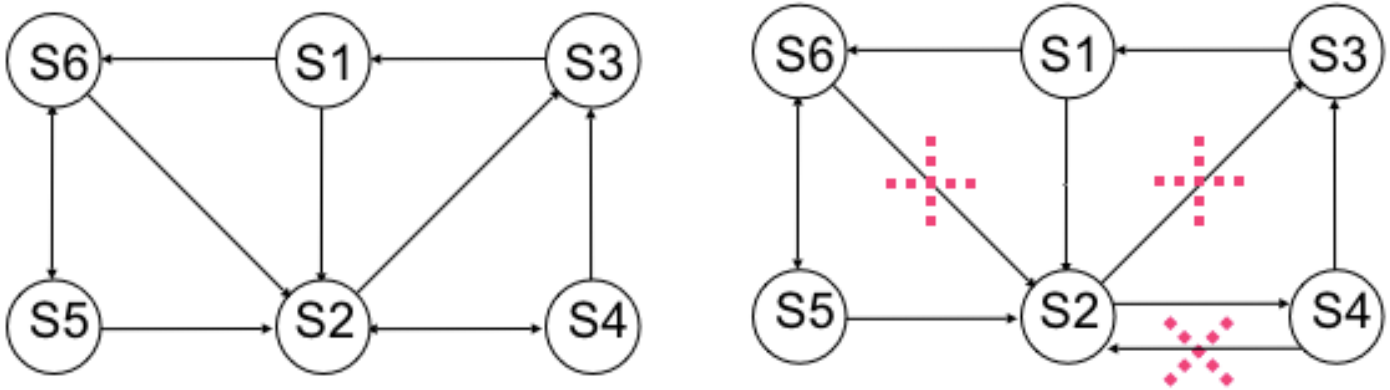
- il existe toutes sortes d'autres consistances (path-consistency, singleton arc-consistency, strong consistencies...)
- ne pas confondre consistance et satsfaisabilité.

Propagation

La propagation d'une contrainte C sur les domaines $D_1 \ldots D_n$ consiste à enlever des domaines toutes les valeurs inconsistantes.

Propagation : exemples

- $X = Y + 3 * Z$ si $X = 10$, $Y = 4$ alors on sait que $Z = 2$,
- $X = Y + 3 * Z$ si $D_Z = \{1..5\}$ et $D_X = \{0..10\}$ alors D_Y peut être intersecté avec $\{-5, 7\}$,
- `alldifferent`(X_1, X_2, X_3) : si on sait que D_1 **et** D_2 valent $\{1, 2\}$, on peut éliminer ces deux valeurs pour X_3.
- pour le cycle de tout-à-l'heure :



Propagation

La propagation **des** contraintes consiste à appliquer la propagation à chaque contraintes jusqu'au point fixe (boucle de propagation).

Remarques

1. Les algorithmes de propagation sont toujours donnés avec une complexité pire des cas. C'est une information complètement inutile (pires des cas simultanés pour toutes les contraintes).

2. Les méthodes de résolution doivent être paramétrées :
   - heuristique de choix de variable,
   - heuristique de choix de valeur,
   - niveau de propagation (choix de la consistance),
   - ordre dans la boucle de propagation,
   - vrai aussi pour les méthodes incomplètes.

   Très souvent, le paramétrage est simplement donné dans les articles.

3. Les propagateurs manipulent des structures de données parfois lourdes (car backtrackables). Parfois ils éliminent des valeurs, parfois non.

Questions

Peut-on prédire si un propagateur est susceptible de servir à quelquechose ?

Peut-on choisir une *bonne* heuristique ?

Peut-on compter / borner les solutions d'une contrainte ?

La programmation par contraintes est efficace. Mais on ne sait pas toujours pourquoi. On a besoin de quantifier des propriétés, sur des objets combinatoires.

Un premier exemple

The alldifferent constraint

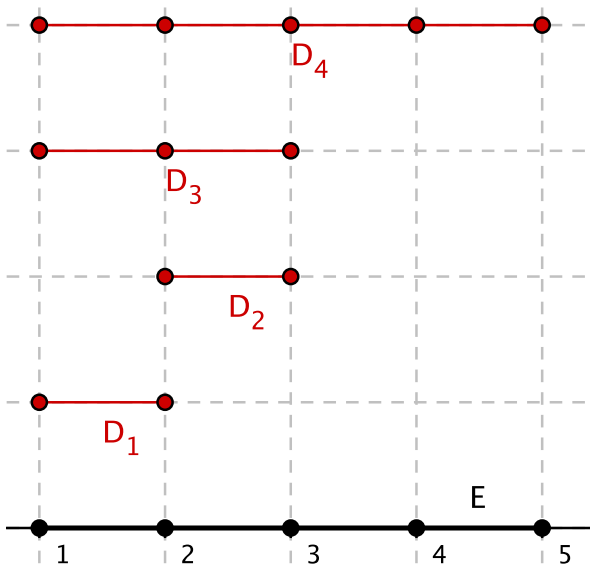
The alldifferent constraint is a well-known global constraint for which many consistency algorithms have been proposed (see Van Hoeve 2001).

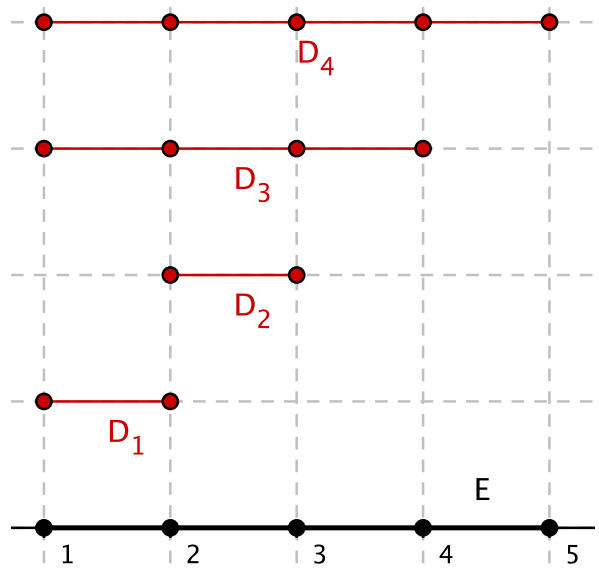
Alldifferent

By definition, `alldifferent`$(V_1, V_2 ... V_n)$ is equivalent to $\forall i \in \{1..n\}, \forall j \neq i, V_i \neq V_j$.

Notations: on notera n le nombre de variables, V_i les variables et D_i le domaine (entier fini) de la variable V_i

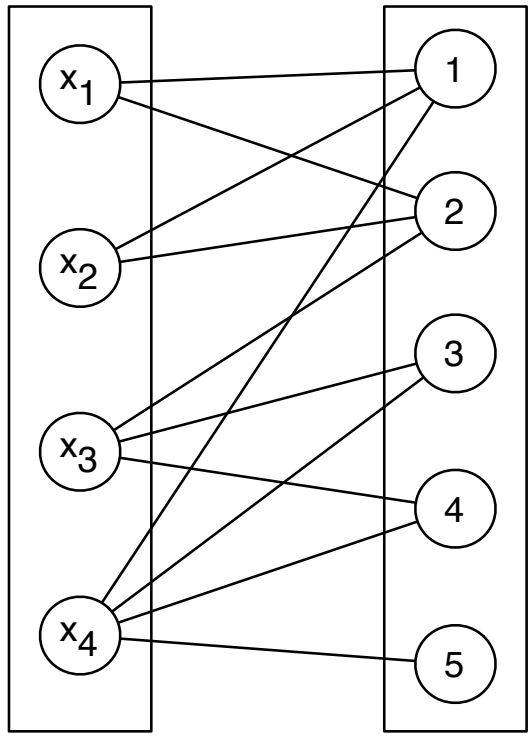
Consistency





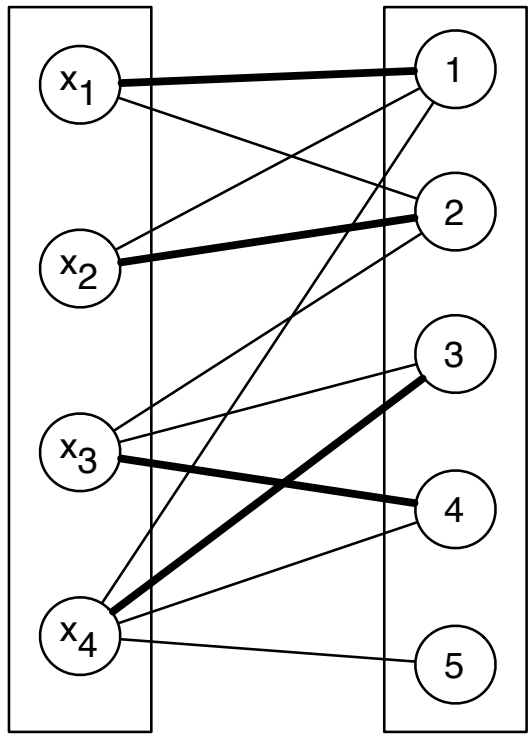
Propagation of `alldifferent` in practice

Bipartite graph with variables and values (inspiré de [Berge 70]).



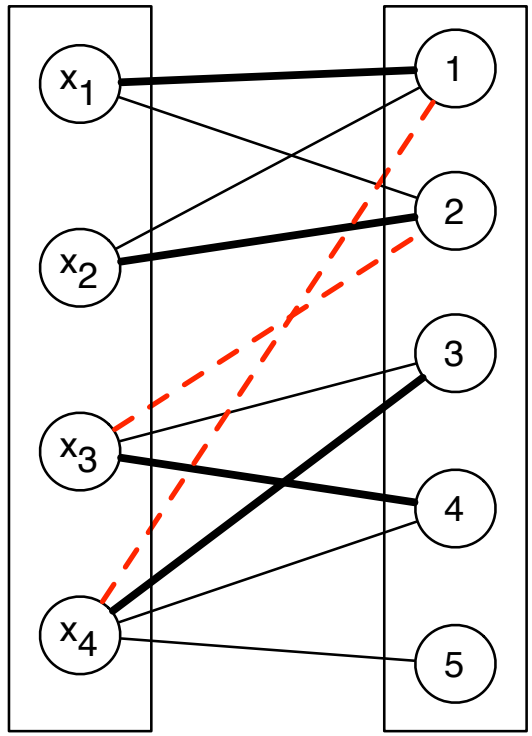
Propagation of `alldifferent` in practice

Assignment satisfying the constraint: pairing of maximum size.



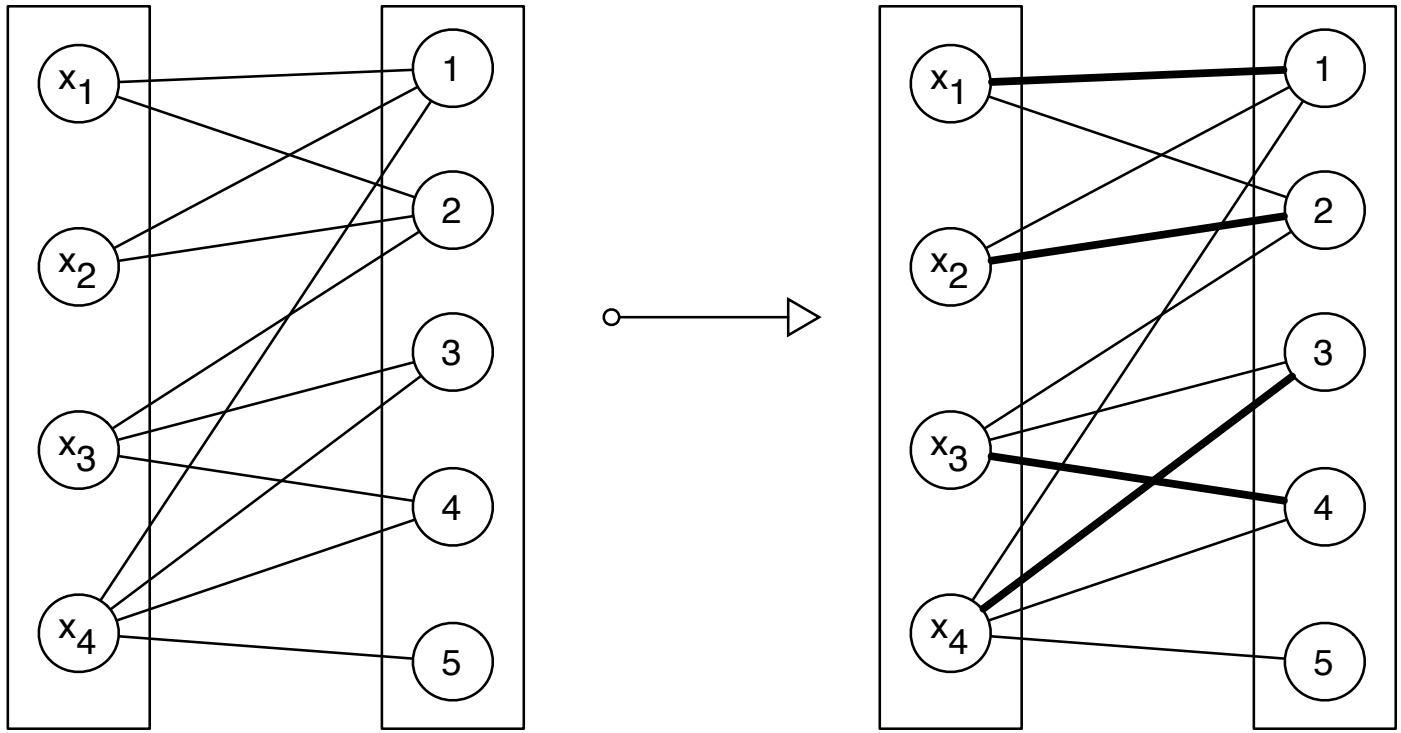
Propagation of `alldifferent` in practice

Edges which do never appear in such a pairing can be removed.



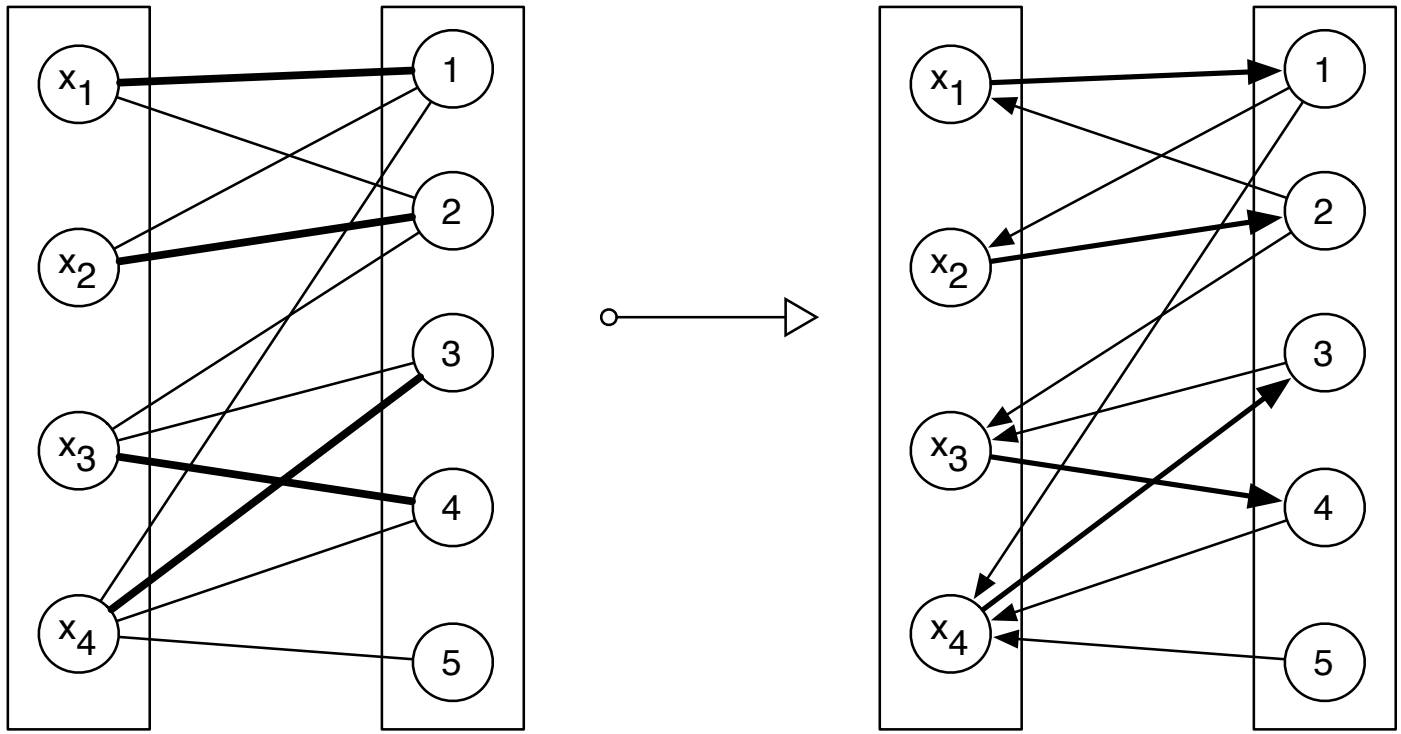
Propagation of `alldifferent` in practice

One can compute a maximum pairing in $O(m\sqrt{n})$



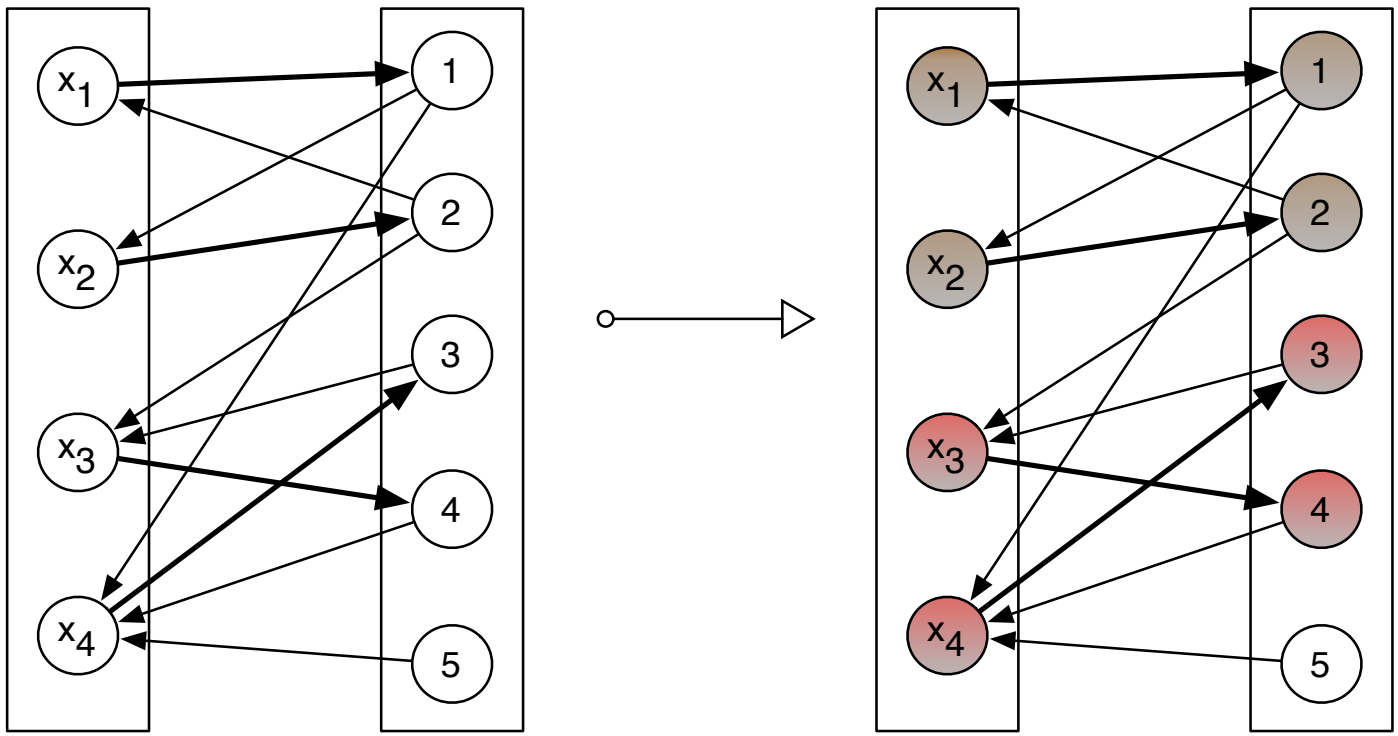
Propagation of `alldifferent` in practice

Transformation into a directed graph G', in $O(m + n)$



Propagation of `alldifferent` in practice

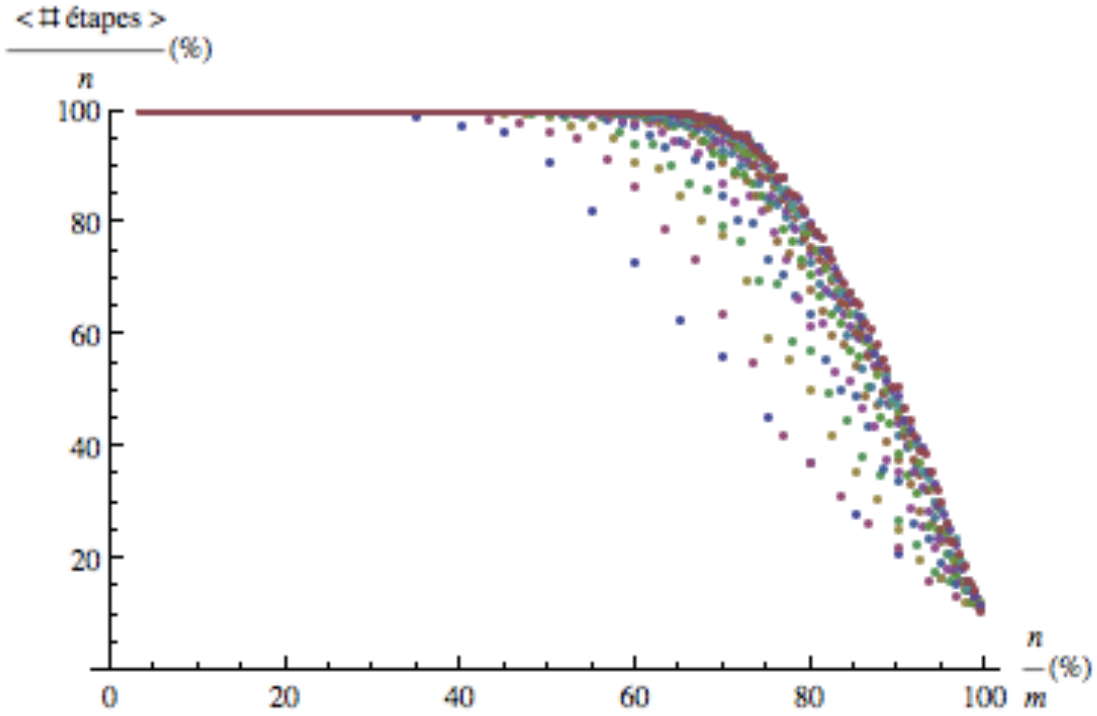
By exploring G' and marking the edges appropriately, we can identify the cfc's in $O(m+n)$, and remove the appropriate edges.



A quoi sert la propagation ?

En théorie : à tout.
En pratique : souvent à rien.

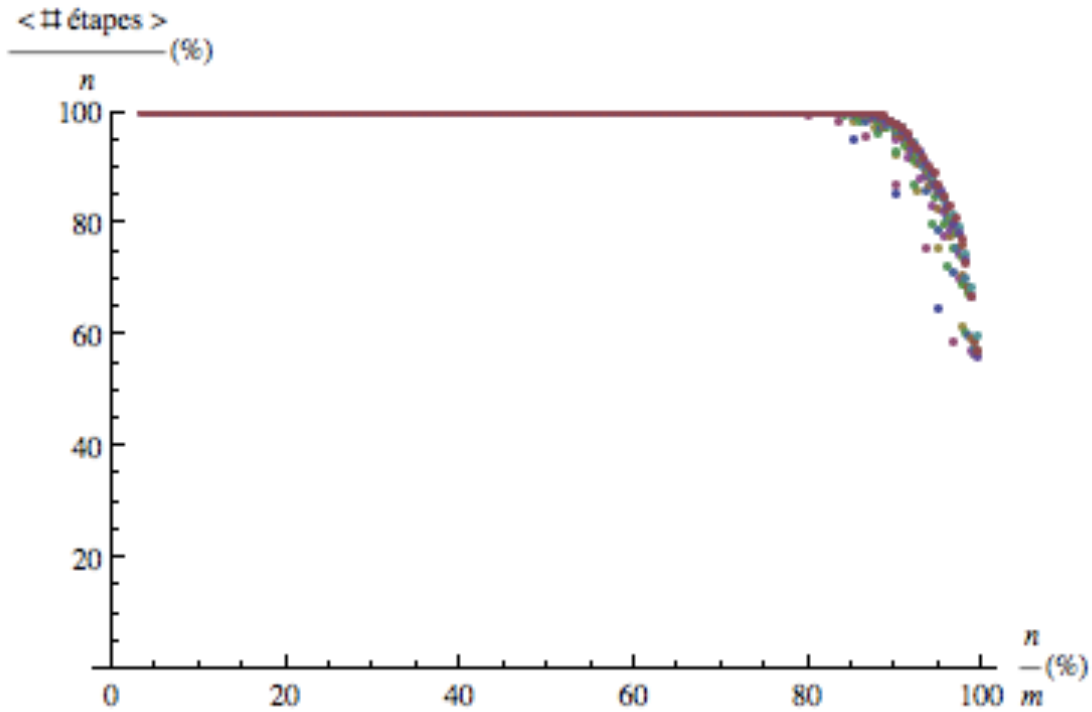
Exemple : Contraint alldifferent (seule) sur des domaines de taille $m/4$, où m est la taille de l'union des domaines.



A quoi sert la propagation ?

En théorie : à tout.
En pratique : souvent à rien.

Exemple : Contraint alldifferent (seule) sur des domaines de taille $3m/4$, où m est la taille de l'union des domaines.



A quoi sert la propagation ?

En théorie : à tout.
En pratique : souvent à rien.

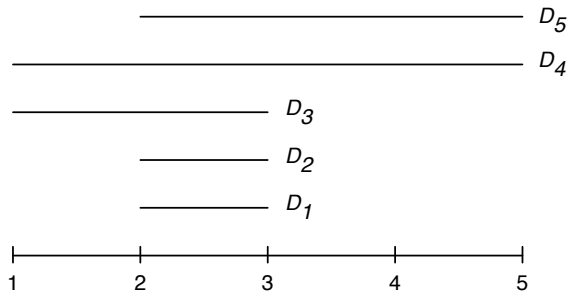
Constat : on appelle à chaque noeud de l'arbre de recherche un algorithme qui, souvent, ne sert à rien, et qui manipule des structures de données lourdes.

NB : ce n'est pas trop grave, car l'algorithme qui propage la consistance est en $O(n * ln(n))$.

Idée

Microscopique

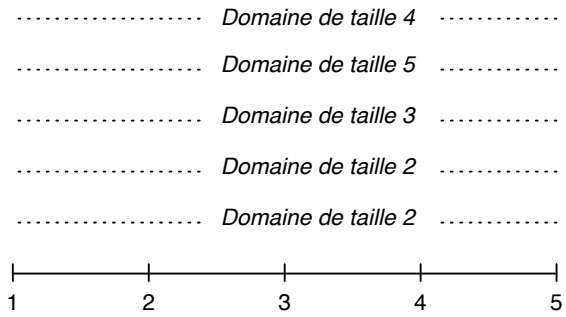
On connaît tout : les domaines, leur placement, etc... Tout est déterminé.



La contrainte est consistante, ou ne l'est pas.

Macroscopique

On ne connaît que certaines caractéristiques : tailles des domaines par exemple.
On probabilise le reste



Quelle est la probabilité pour que la contrainte soit consistante ?

Modèle probabiliste pour la consistante de bornes

Sont connus :

- le nombre de variables n,
- les tailles des domaines $d_1 ... d_n$,
- l'union des domaines $E = \cup_{1 \leq i \leq n} D_i$,
- la taille de l'union des domaines $m = |E|$.

NB on peut supposer sans perte de généralité que E est un intervalle.

Domaines probabilisés

On suppose que chaque domaine D_i varie uniformément sur E, indépendamment des autres domaines. On a donc :
$P(D_i \text{ est à une position fixée}) = \frac{1}{l - d_i + 1}$

Consistance de bornes

Definition

Soit I un intervalle inclus dans E. On note $K_I = \{i, D_i \subset I\}$ l'ensemble des indices des domaines complètement inclus dans I.

Definition

Un intervalle I de E est dit *de Hall* ssi $|K_I| = |I|$.

Du point de vue du `alldifferent`, les intervalles de Hall consomment toutes les valeur de E qu'ils contiennent.

Proposition

Une contrainte `alldifferent`est bornes-consistante ssi pour tout intervalle $I \subset E$,

- (i) $|K_I| < |I|$,
- (ii) ou bien $|K_I| = |I|$ et $\forall j \notin I, I \cap D_j = \emptyset$.

On peut calculer séparément les probabilités de (i) et de (ii).

Incremental consistency

Assume that V_n, which is assigned a value $x \in D_n$. For the other variables:

- If $x \notin D_i$, then the domain remains unchanged,
- if $x \in D_i$, the domain D_i is now the union of the two disjoint intervals $[\underline{D_i}...x-1]$ and $[x+1...\overline{D_i}]$.

Definition

The `alldifferent` constraint *remains BC* after the instanciation of V_n iff `alldifferent` is BC with respect to the new domains $D'_1...D'_{n-1}$.

Incremental consistency

Proposition

With the above notations, the `alldifferent`constraint remains BC after the instanciation of V_n to a value x iff for all $I' \subset E'$, such that $I = I' \cup \{x\}$ and $D_n \not\subset I$, none of the two following statement holds:

(i) $|K_I| = |I|$,

(ii) $|K_I| = |I| - 1$ and there exists $i \notin K_I$ such that $\underline{D_i} \in I$ or $\overline{D_i} \in I$.

Main result

Proposition

For a given interval $I \subset E$ and a domain $\mathcal{D}$ drawn with a uniform distribution on $\mathcal{I}(E)$, with $m = |E|$ and $l = |I|$, let p_l and q_l be the respective probabilities that $\mathcal{D} \subset I$, and that either $\mathcal{D} \cap I = \emptyset$, or $\underline{\mathcal{D}} < \underline{I} < \overline{I} < \overline{\mathcal{D}}$. Then

$$p_l = \frac{l(l-1)}{m(m-1)}; \qquad q_l = \frac{(m-l)(m-l-1)}{m(m-1)}.$$

Main result

Theorem

Consider an `alldifferent` *constraint on variables* $V_1, ..., V_n$, *initially BC. Let* E *and the domains* $\mathcal{D}_i$, $1 \leq i < n$, *satisfy the assumptions* **A1** *and* **A2**. *Furthermore, assume that we know the domain* $D_n = [a...b]$. *Then the probability* $P_{m,n,x,a,b}$ *that the constraint remains BC after the instanciation of the variable* V_n *to a value* x *is*

$$P_{m,n,x,a,b} = \prod_{l=1}^{n-2} (1 - P^{(1)}_{m,n,l} - P^{(2)}_{m,n,l})^{\Phi(m,l,x,a,b)}$$

where $\Phi(m, l, x, a, b)$ *is defined as*
$\min(x, m-l) - \max(1, x-l) + 1$, *if* $l < b - a$ *and as*
$\min(x, m-l) - \max(1, x-l) - \min(a, m-l) + \max(1, b-l)$,
otherwise,
and where

$$P^{(1)}_{m,n,l} = \binom{n-1}{l+1} p_{l+1}^{l+1} (1 - p_{l+1})^{n-l-2},$$

$$P^{(2)}_{m,n,l} = \binom{n-1}{l} p_{l+1}^{l} \left((1 - p_{l+1})^{n-l-1} - q_{l+1}^{n-l-1} \right),$$

Main result

Theorem

Define a function $\Psi(m, x, a, b)$ *for* $1 \leq a \leq x \leq b \leq m$ *and* $a \neq b$ *by*

- $\Psi(m, x, a, a+1) = 1$, *except*
  $\Psi(m, 1, 1, 2) = \Psi(m, m, m-1, m) = 0$;
- *If* $b > a+1$ *then* $\Psi(m, x, a, b) = 2$, *except*
  $\Psi(m, 1, 1, b) = \Psi(m, m, a, m) = 1$.

Then the probability $P_{m,n,x,a,b}$ *that the constraint remains BC after the instanciation of* V_n *to the value* x *has asymptotic value*

- *if* $n = \rho m,\ \rho < 1$, $1 - \Psi(m, x, a, b)\frac{2\rho(1-e^{-4\rho})}{m} + O\left(\frac{1}{m^2}\right)$;
- *if* $n = m - i,\ i = o(m)$,
  $e^{C_i}\left(1 - \Psi(m, x, a, b)\frac{2(1-e^{-4})+D_i}{m} + O\left(\frac{1}{m^2}\right)\right)$.

[...]

Main result, relifted

Key result: identification of two different asymptotical regimes for the probability of remaining consistent after an instanciation.

- if $n/m = \rho, \rho < 1$,
  then the limit is 1,

  the BC propagation is useless
- if $n = m - o(m)$,
  then the limit is strictly smaller than 1.

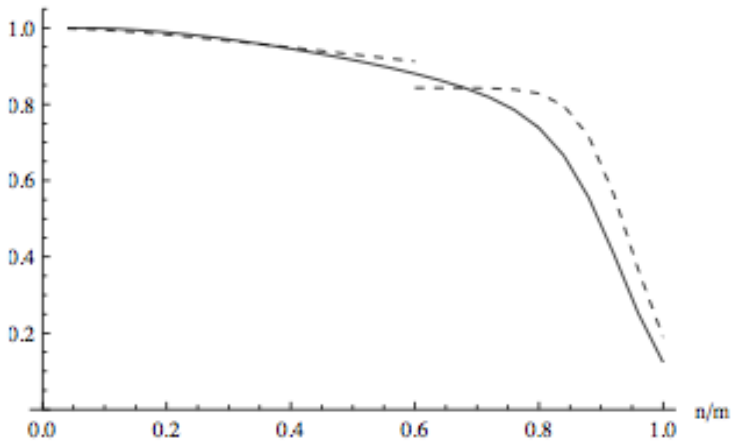
  it depends

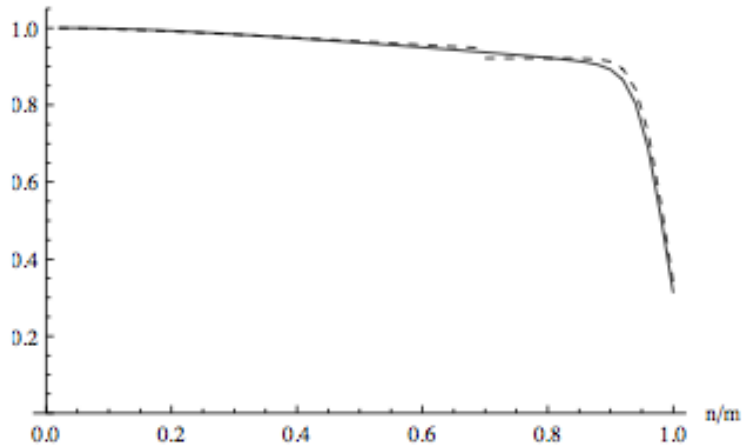
More in [du Boisberranger, Gardy, Lorca, Truchet, Analco 2013].

Qu'en conclure ?

- On a un indicateur probabiliste pour la consistance d'un `alldifferent`.
- Plus intéressant, on identifie deux comportements pour la contraintes:
  - si n est constamment inférieur à m, la probabilité de rester consistante après instanciation tend vers 1,
  - si n est proche de m, elle tend vers une constante strictement plus petite que 1.

Exemple





Numerical evaluation of the theoretical (plain) and approached (dashed) probability $P_{25,n,15,3,19}$ for $m = 25$ (top) or 50 (bottom), and for n varying from 1 to m.

Et maintenant ?

- difficile à utiliser en pratique,
- **mais** plus d'espoir pour les autres contraintes de cardinalité : modèles d'urnes (avec Danièle Gardy), modèles à base de graphes (avec Giovanni Lo Bianco, Vlady Ravelomanana).

Un autre exemple

Contexte

On considère

- un problème NP sur une donnée de taille k,
- un algorithme qui résout ce problème, avec un temps de calcul en $O(e^k)$.

Question : est-ce intéressant de paralléliser l'algorithme ?

- Non : on sera toujours en $O(e^k)$
- Oui : pour un temps donné, on espère résoudre pour de plus grands k, et de toutes façons *les ressources sont disponibles*.

Notion de speedup : temps gagné à paralléliser.

Local Search

Focus on a specific family of Las Vegas algorithms: Local Search (LS). Basic behavior:

- starts randomly on the search space,
- iteratively repeats:
  - explore a small neighborhood of the current configuration,
  - find the best point of the neighborhood w.r.t. the function to optimize,
  - move to this best point,
- adds higher-level mechanisms to avoid cycling in local optima (short-term memory, randomization, etc).

NB : LS includes several random components $\rightarrow$ Random computation time.

Parallelization

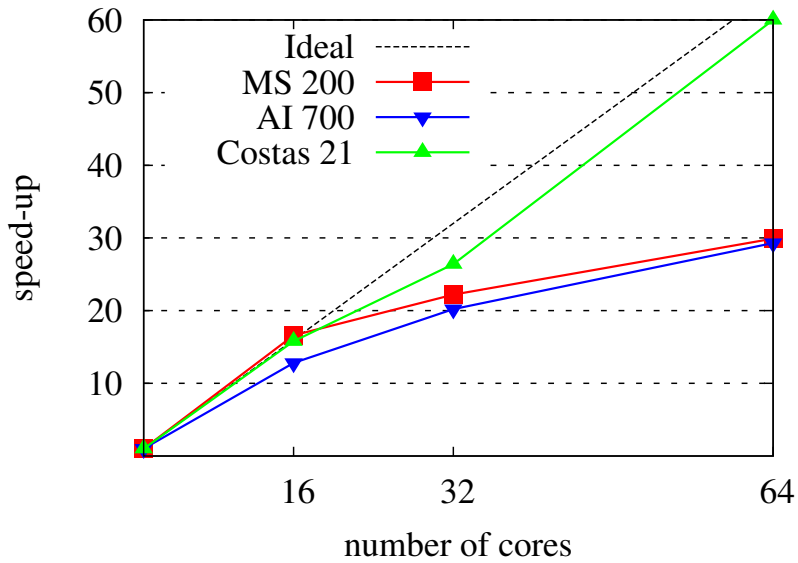
Studied since the early 1990s [Verhoeven-Aarts'95], with a recent interest [Alba'04,Crainic-Toulouse'02...]

Classification:

- single walk methods: parallelize some components inside a single LS algorithm (neighborhood exploration),
- multiwalk methods:
  - with shared memory: solutions pools [Crainic-et-al'04],
  - no shared memory: portfolio algorithms (SAT), multiwalk (Local Search) extensions

Motivation

We tried to parallelize with more or less naive methods, and observed:



Questions:

- why so different speed-ups ?
- can we predict these behaviors ?
- can we at least understand them ?

Multiwalk extension

Definition (Multi-walk Las Vegas Algorithm)

The multiwalk extension A' of a LS algorithm A consists in n copies of A running independently. The first copy that terminates kills the other, and outputs its solution.

Remarks:

- if A always terminate, so does A',
- there is no shared memory,
- very easy to implement and study: *a canonical test-case for parallel combinatorial optimization.*

Notations

Sequential runtime

Let Y the random variable corresponding to the sequential runtime, with distribution f_Y and cumulative distribution $\mathcal{F}_Y$.

- by def: $\mathcal{F}_Y(x) = \mathbb{P}[Y \leq x]$
- extension to the reals: $\mathcal{F}_Y : \mathbb{R} \to \mathbb{R}$

Parallel runtime

Algorithms

Given: a LV sequential program.

The program is run on n processors.

The first run which terminates kills the others.

Runtime ?

Probabilities

Given: a probability distribution.

Let $X_1 ... X_n$ n i.i.d. random draws, each following distribution f_Y.

Let $Z^{(n)} = \min(X_1 ... X_n)$.

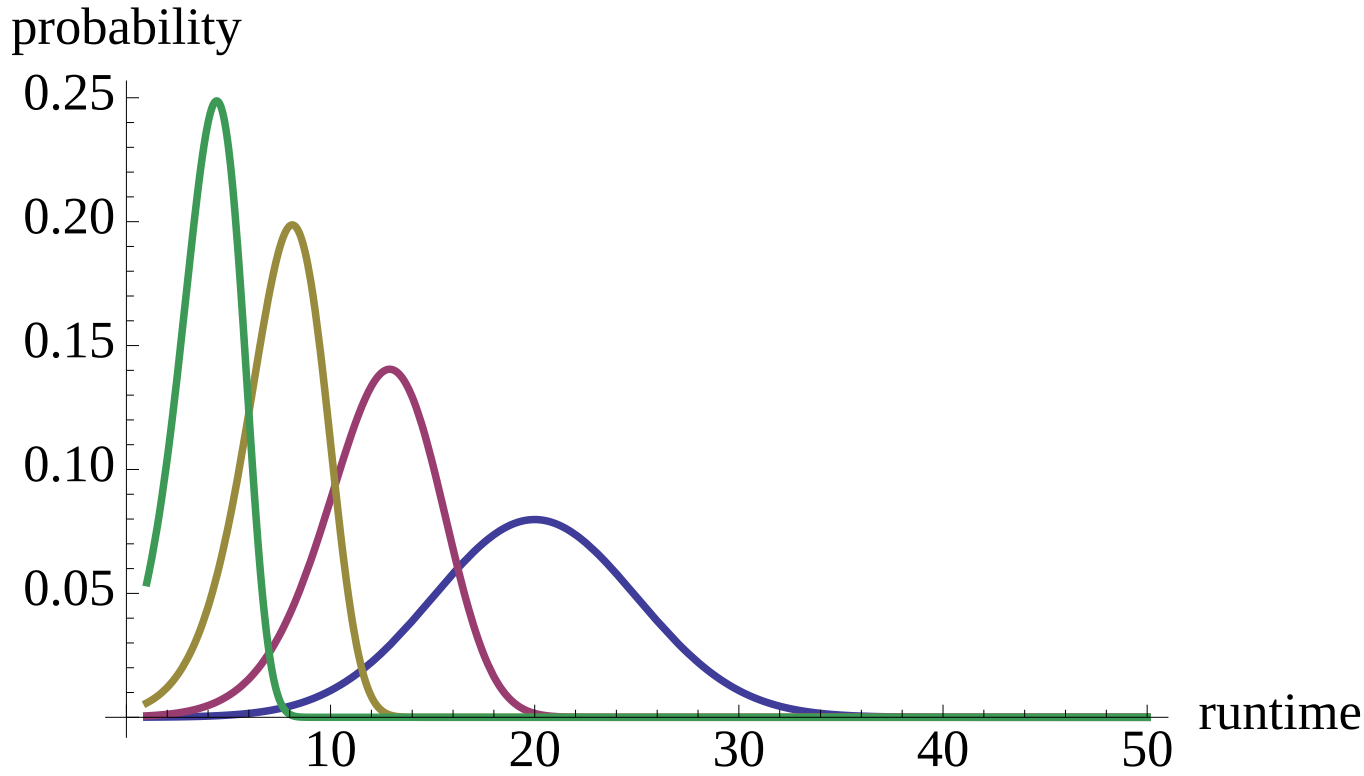
$f_{Z^{(n)}}$?

Generic expression of $Z^{(n)}$

Remark: related to *order statistics*, *i.e.* the statistics of ordered random variables [David-Nagaraja'03].

$$\begin{aligned}
\mathcal{F}_{Z^{(n)}} &= \mathbb{P}[Z^{(n)} \leq x] \\
&= \mathbb{P}[\exists i \in \{1...n\}, X_i \leq x] \\
&= 1 - \mathbb{P}[\forall i \in \{1...n\}, X_i > x] \\
&= 1 - \prod_{i=1}^{n} \mathbb{P}[X_i > x] \\
&= 1 - (1 - \mathcal{F}_Y(x))^n
\end{aligned}$$

Example: gaussian distribution



Blue: f_Y
Pink, yellow, green: $f_{Z^{(n)}}$ for $n = 10, 100, 1000$

Speed-up

Definition (Speed-up)

$\mathbb{E}[Y]/\mathbb{E}[Z^{(n)}]$.

Explicit formula:

$$\begin{aligned}\mathbb{E}[Z^{(n)}] &= \int_0^\infty t f_{Z^{(n)}}(t)\,dt \\ &= n\int_0^\infty t f_Y(t)(1-\mathcal{F}_Y(t))^{n-1}\,dt\end{aligned}$$

An easy case: exponential distribution

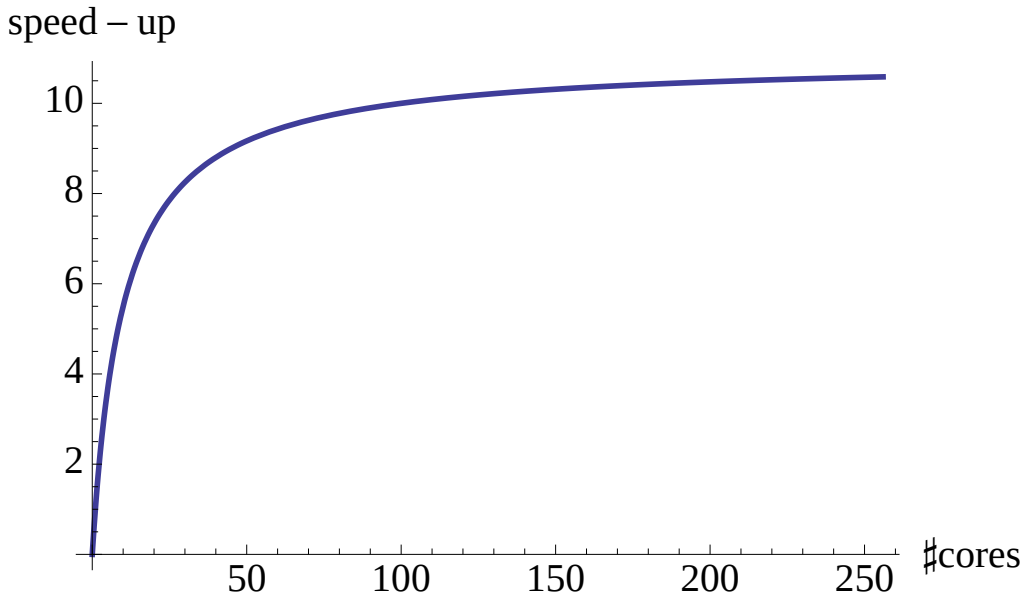
Hypothesis: the sequential runtime admits an exponential distribution (as suggested by [Aiex-et-al'02]). We generalize this to a shifted distribution:

$$f_Y(t) = \begin{cases} 0 & \text{if } t \leq x_0 \\ \lambda e^{-\lambda(t-x_0)} & \text{if } t > x_0 \end{cases}$$

Then the above formulas can be symbolically computed by hand and yields:

$$\mathcal{G}_n = \frac{x_0 + \frac{1}{\lambda}}{x_0 + \frac{1}{n\lambda}}$$

An easy case: exponential distribution

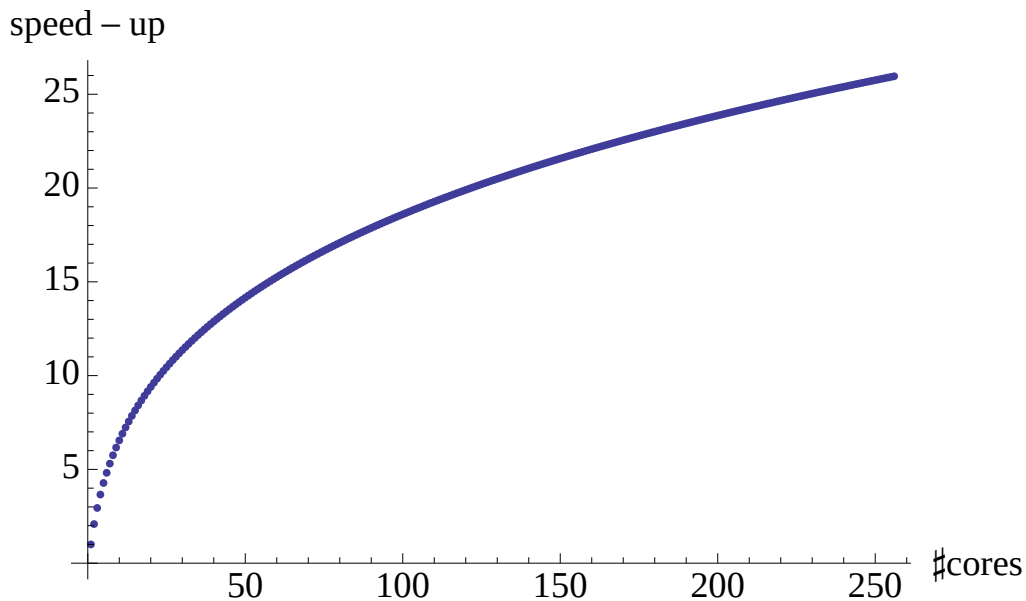


Predicted speed-up in case of an exponential distribution.
Linear if $x_0 = 0$ as shown by [Verhoeven-Aarts'95].
Admits a finite limit $1 + \frac{1}{x_0 \lambda}$ if $x_0 > 0$.

A tractable case: lognormal distribution

Hypothesis: the sequential runtime admits a lognormal distribution.

Recent works in statistics give an explicit formula for the minimum distribution [Nadarajah'08].



Predicted speed-up in case of a lognormal distribution.

Summary

- If we have a symbolic formula for the sequential distribution, we have a symbolic formula for the speed-up.
- If the sequential distribution is one of those for which the min distribution is known, then we have a numerical estimation of the speed-up.
- In the field of order statistics, the min distribution has been studied for several classical distributions: exponential, gaussian, lognormal, gamma, beta... [Nadarajah'08]

Question: can we approximate the sequential runtime with such a distribution ?

Benchmark: 3 series of problems

- classical problems (all-interval series, costas arrays, magic squares) with adaptive search [Codognet-Diaz'01]
- SAT problems, 10 crafted and 10 random, with local search (CCASAT and Sparrow)
- classical problems with a complete solver (with a random value selection heuristic)

Methodology

For each problem:

- run the sequential algorithm many times to obtain an observation of f_Y.
- approximate f_Y by a known theoretical distribution (statistical test).
- compute the theoretical speed-up.
- run the parallel algorithms to obtain experimental speed-ups.

Experiments have been made on Grid'5000 (Griffon with 736 cores), with freeware implementations of each methods (C libraries).

Approximation of the experimental runtime

On a large number of runs (around 800) with adaptive search, we have

- for the All-Interval Series problem, with $n = 700$: a shifted exponential distribution, with $x_0 = 1217 > 0$.
- for the Magic square with $n = 200$: a shifted lognormal distribution, with $x_0 = 6210 > 0$.
- for the Costas array with $n = 21$: a *non-shifted* exponential distribution, with $x_0 = 0$.

Conclusion

A generic method to estimate speed-ups, provided that the sequential runtime is known accurately enough.
Works with any Las Vegas algorithm.

- Need to better understand the role of x_0
  A key point is probably the observed value of x_0 for a limited number of runs ($<< \infty$) compared to the average runtime.
- Good accuracy on the considered benchmark: does it scale ?
  - test on small instances,
  - machine-learning on large sets of instances [Arbelaez, Truchet, O'Sullivan, ICTAI 2016]

More in [Truchet, Arbelaez, Richoux, Codognet, J. of Heuristics 2016]

Leçons chèrement apprises

Il est évident que la programmation par contraintes, et sans doute d'autres familles d'optimisation combinatoires, ont des gros besoins d'analyse en moyenne.

Quelques difficultés:

- parfois, les belles méthodes (comme la combinatoire analytique) ne s'appliquent pas,
- les modèles sont nécessairement approximatifs (CSP random, SAT random),
- il est difficile d'atteindre quelquechose d'efficace.

Des tas de questions !

- étudier les contraintes de cardinalité dures : `nvalue`
- compter les solutions pour des contraintes de cardinalité
  déjà fait pour `alldiff` [Zanarini, Pesant, CPAIOR 2010], en cours pour `gcc` (Lo Bianco, Lorca, Truchet, Pesant), à faire pour `nvalue`
- trouver des transitions de phase pour la satisfaisabilité,
  en cours pour `gcc` (Lo Bianco, Lorca, Ravelomanana, Truchet)
- étudier les temps d'exécution des algorithmes randomisés sur des gros paquets d'instances
- ...

Conclusion

La programmation par contraintes a besoin d'analyse en moyenne !

Ne pas oublier de faire un peu de pub pour Binaire et "Il était une fois... ma thèse"...